Blueprint for a Computer Immune System

by Jeffrey O. Kephart, Gregory B. Sorkin, Morton Swimmer, and Steve R. White
IBM Thomas J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598

This paper was originally presented at the Virus Bulletin International Conference in San Francisco, California, October 1-3, 1997.

Abstract

There is legitimate concern that, within the next few years, the Internet will provide a fertile medium for new breeds of computer viruses capable of spreading orders of magnitude faster than today's viruses. To counter this threat, we have developed an immune system for computers that senses the presence of a previously unknown pathogen, and within minutes automatically derives and deploys a prescription for detecting and removing it. The system is being integrated with a commercial anti-virus product, IBM AntiVirus, and will be available as a pilot in 1997.

1. Introduction

During the last decade, computer virus authors and anti-virus technologists have been engaged in an escalating arms race. Today, these forces are roughly in balance: computer viruses are a nuisance, but a manageable one. When a new virus appears, prescriptions for detecting and removing it are developed by human experts. This information is incorporated into the data files of anti-virus products, and is eventually distributed to end users. The time scale on which viruses spread globally is generally several months, commensurate with the time scale on which users typically receive updates.

However, the explosive growth of the Internet and the rapid emergence of applications that disregard the traditional boundaries between computers threaten to increase the global spread rate of computer viruses by several orders of magnitude. Traditionally, a PC virus has had to rely on (unwitting) human intermediaries to help it spread within a PC or from one PC to another: a human has had to execute an infected program, and to copy it from one machine to another, e.g. via diskette or ftp. But the tasks of executing and copying are becoming automated to an ever greater extent. The problem is exacerbated further by the fact that objects like documents and spreadsheets, which tend to be transferred between computers at a much greater rate than executable programs, are now increasingly likely to harbor code themselves in the form of macros, and these macros may contain viruses.

For example, modern-day mail programs permit text documents or spreadsheets to be sent very simply as e-mail attachments. When opened by the recipient, the type of the attachment can be recognized, and an application capable of interpreting and displaying it can be launched automatically. When the application loads the attachment, any macros contained therein may be executed, giving any viral macros the opportunity to spread. Note that, in this scenario, the virus' dependence upon human intervention has been reduced substantially from what it has been traditionally. The sender can now (unwittingly) transmit an infected object by attaching it to e-mail with a few mouse clicks, one last mouse click sending it on its way.

Execution is even easier: unsophisticated recipients may not even realize that, simply by opening their mail, they are automatically executing applications that may in turn automatically execute macros.

Complete automation of the cycle of transmitting a virus from computer A to computer B and then executing it on computer B cannot be far off. No great technological breakthroughs are required. Already, a number of mail programs permit the user to create agents that automatically send or process mail on behalf of the user. One can expect that, soon, viruses will exploit these capabilities, and once they do they will no longer be held back by the stumblings and bumblings of humans. Other vehicles for fully automated replication will be available soon. A few years from now, the Internet may seethe with millions or billions of mobile agents, which could serve as a fertile medium for a new type of virus that could direct its host to quickly visit a large number of sites and infect other agents there. Viruses such as these will be in control of their own destiny, replicating at rates that may be checked only by network latency and processing speed.

In short, the nature of computer viruses and their ability to propagate is on the cusp of a fundamental, qualitative change -- one that demands an equally fundamental change in the way we must defend against them. Epidemiological modeling and common sense both dictate that eliminating a virus requires killing it off at a rate faster than it can spread. With humans completely removed from the replication loop, the only way to respond quickly enough to new viruses will be to remove humans from the response loop as well.

But how? Fortunately, nature supplies us with a ready answer. Higher organisms' immune system are effective mechanisms for protecting against rapidly spreading, rapidly evolving pathogens such as bacteria and viruses. Inspired by the analogy between computer viruses and biological ones, we have designed and prototyped an "immune system for cyberspace" that is capable of automatically deriving and distributing anti-virus data within minutes of a new virus's first detection.

An exploration of the connections between our system and biological immune systems can be found in [1,2]. This paper emphasizes issues pertaining to the design and implementation of a robust, practical, industrial-strength immune system. Section 2 discusses the minimal criteria that ought to be satisfied by such a system. Then, section 3 elaborates some implementation details that illustrate how we have met these criteria in the IBM immune system design and prototype.

2. Requirements for a computer immune system

In order to describe what we mean by a computer immune system, we shall first describe briefly some of the salient features of biological immune systems, using this as a basis for a broad outline of the function of a computer immune system. Then, we shall define some minimal practical requirements that must be satisfied by any commercially viable computer immune system.

When first exposed to a pathogen, the immune system of vertebrates and many non-vertebrates reacts in a non-adaptive, non-specific way. This "innate" immune system provides a first line of defense, and, in vertebrates, also triggers a more adaptive, specific immune response. To discriminate pathogens from benign entities, the innate immune system combines knowledge of the self (e.g. proteins that are present on self-cells, but usually not on foreign cells [3]) with general knowledge of broad classes of harmful entities (signaled by double-strand RNA, which occurs in much greater concentrations in certain classes of viruses than in mammalian cells, or by the presence of a peptide produced copiously by bacteria but only in minute amounts by mammals [4]). The innate immune system can also produce a generic response to pathogens which disables or kills them.

In addition to the innate immune system, vertebrates also possess a more sophisticated, adaptive immune system whose components include antibodies, B-cells, and T-cells. When a pathogen is first encountered, only a few antibodies or immune cell receptors may bind weakly to the pathogen or fragments of it, but over the course of a few days an adaptive process produces numerous immune cells and antibodies that bind strongly enough to detect and disable the pathogen in a very precise way. Henceforth, the immune system will retain a memory of the pathogen, and be armed against repeat attacks by it [5].

A computer immune system must include components patterned on both the innate and the adaptive immune systems. Like the innate immune system it must sense pathogenic anomalies in a generic way, but by itself this is not enough. A computer immune system must also, like the adaptive immune system, develop ways of identifying pathogens very specifically, and then it must use this precise identification to purge them.

Combining these very broad statements with a number of practical considerations, we set forth some criteria that must be met to provide real-world, functional protection from rapidly spreading viruses:

- **Innate immunity.** The system must be capable of detecting the presence of a high proportion of viruses that are unknown to it specifically. The system should recognize previously unknown viruses of all types; currently, this would include file infectors, boot-sector infectors, and macro viruses. It is particularly important that the system be geared towards detecting viruses that are found in real incidents. For example, less emphasis need be placed on detecting overwriting viruses, since these are never prevalent.

- **Adaptive immunity.** The system must be capable of automatically deriving from a single sample a "prescription" for detecting and, if possible, removing all instances of the virus.

- **Delivery and dissemination.** The system must deliver the prescription to the afflicted computer, and must also facilitate dissemination of the prescription to other machines both in its neighborhood and around the world.

- **Speed.** The system must be capable of deriving and disseminating prescriptions on a time scale faster than the virus can spread. Currently, a reasonable goal is to derive and deliver a prescription to the originally afflicted machine in less than ten minutes from when an anomaly is first detected, and to permit dissemination of the prescription to neighboring machines within less than half an hour. The system should be capable of protecting anyone from any newly discovered virus within a day at most. These requirements will become even stricter as technology permits computer viruses to spread faster.

- **Scalability:**

  Analysis. If a new, particular virulent virus begins to spread quickly, many machines at one site and/or at different sites may experience the virus simultaneously. The system must be capable of very high throughput, handling at least thousands of simultaneous requests for virus analysis.

Updates. The system must be able to update tens of millions of PCs on at least a daily basis.

- **Safety and reliability.** The prescription produced by the immune system must be sufficiently reliable that it can be deployed worldwide without being checked by humans. Preferably the false positive rate should be at least as low and the removal information at least as accurate as is typically achieved by expert humans.
- **Security.** Virus samples must be protected in such a way that they cannot be intercepted and read by a third party while in transit. Even more importantly, appropriate measures must be taken to ensure that any prescriptions employed by a client machine are uncorrupted, genuine copies of those produced by the immune system.
- **Customer control.** Customers must be able to control the conditions under which virus samples and other information are sent out of their enterprise. They must likewise be able to control the conditions under which prescriptions are received and disseminated to the originally infected client and possibly other machines. Some customers may want manual control in one or both directions, others may want the speed given by completely automated response.

Most of today's major anti-virus products possess a few bits and pieces from among these ingredients. Most use heuristics of various sorts to recognize previously unknown viruses. Some anti-virus vendors even use automated procedures to assist with analyzing viruses. However, designing and implementing a robust defense against rapidly-spreading viruses is a serious undertaking, and requires meeting all of the above criteria. No commercial anti-virus offering comes anywhere close to offering this level of defense today.

The remainder of this paper describes our efforts in this direction: the design and prototype of the IBM immune system for cyberspace, which is intended to defend millions of real customers against real viruses.

3. Implementing an Immune System for Cyberspace

In broad terms, our implementation of the immune system consists of the steps of:

1. Discovering a previously unknown virus on a user's computer.
2. Capturing a sample of the virus and sending it to a central computer.
3. Analyzing the virus automatically to derive a prescription for detecting and removing it from any host object.
4. Delivering the prescription to the user's computer, incorporating it into the anti-virus data files, and running the anti-virus product to detect and remove all occurrences of the virus.

5. Disseminating the prescription to other computers in the user's locale and to the rest of the world.

We shall now describe each of these steps in further detail, illustrating along the way how the criteria set forth in the previous section are met.

3.1 Discovering previously unknown viruses: the innate immune system

To be effective, an immune system must do a very good job of sensing previously unknown viruses that are present in the computer system that it is protecting. Like the "innate" immune system, the computer immune system can combine self-knowledge (embodied for example in change detection and change analysis) with generic knowledge of broad classes of potential pathogens (embodied for example in static analysis based on viral machine-code features). These and other heuristics, some based on dynamic, behavior-based analysis, can be incorporated in a very natural way into an existing anti-virus product running on user's PCs. Here we mention some of the main heuristics used to discover previously unknown file infectors, boot-sector infectors, and macro viruses, all of which are either already integrated into IBM AntiVirus, or will be shortly.

3.1.1 File infectors

To catch previously unknown file-infecting viruses, we use two main heuristics. The first is based on a "generic disinfection" technique [6]; the second on a classifier that recognizes potential viruses based on characteristic machine-code features [2,7].

Generic disinfection heuristic The underlying principle of the generic disinfection heuristic is that a program F' infected with a virus can generally be restored to its original, uninfected version F . The reason for this is simple. After a virus has carried out its own function, the best way for it to avoid immediate detection is to allow its host program to continue normally. This generally means that the virus must be able to reconstruct the host in its original form, which in turn requires that none of the original host bytes be destroyed. In other words, for non-overwriting viruses, F' can be regarded as a reversible transformation of F , i.e. the original host F is in principle reconstructible from F' . On the other hand, legitimate changes such as updating a program to a new version are unlikely to be similarly reversible: too much will change, and some bytes that were contained in F will no longer be present in F' .

Therefore, if a program has been modified from F to F' and the generic disinfection algorithm could recover F from F' , it may be concluded with a high degree of confidence that F' contains a virus. If so, the user is given the option to carry out the disinfection, and a sample of the program is captured.

Generic disinfection algorithm As employed in IBM AntiVirus, the generic disinfection algorithm works as follows. When the product is first installed, a fingerprint of each executable file is computed and stored in a database. The fingerprint consists of less than 100 bytes of information about each program F , including its size, date, and various checksums. On subsequent scans of the system, the fingerprint is recomputed and compared with the stored one. If it has changed, the new version of the program, F' , will be scanned for known viruses. If the change cannot be attributed to any known virus, the generic disinfection algorithm tries to reconstruct the original program F from the new version F' plus the fingerprint.

The reconstruction is attempted roughly as follows. First, some of the fingerprint data are used to locate the beginning and end of F within F' , say at offsets b and e respectively. Then, if e - b = |F|, the length of F , the checksum of the bytes between b and e is computed. If the checksum matches the original checksum contained in the fingerprint, the block of bytes between b and e is taken to be an exact reconstruction of F . If e - b > |F| or e < b, then a series of trial reconstructions is attempted, in which a block F1 beginning at b is concatenated with a block F2 ending at e, with |F1| + |F2| = |F|. If the checksum of a reconstruction matches the original checksum, the reconstruction is assumed to be correct.

Generic disinfection implementation The number of candidate reconstructions of F can be of the same order as its length. Since computing a single checksum requires an amount of computation proportional to |F|, a naive implementation of the generic disinfection algorithm would require running time quadratic in the file length. In practical terms, it would take several days to reconstruct an average executable on an average PC. Fortunately, for CRC checksums (cyclic redundancy check [8]), there is a very efficient implementation (modeled on the Karp-Rabin hash for string matching [9]) allowing all the checksums together to be computed in linear time, and in a fraction of a second on a PC.

To see how this it works, suppose e - b > |F|. In this case, our working assumption is that the original file has been split at an unknown point into blocks F1 and F2, with a block of virus bytes introduced between them. A first candidate reconstruction hypothesizes that the split occurred as close before e as is consistent with our recognition of e as the endpoint of the original file F ; this is checked by computing the checksum in the standard way. Then, successive candidates are generated by extending the beginning of the block F2 backwards one byte at a time, with a corresponding backwards shrinkage from the end of block F1.

The key point is that the checksum of each candidate need not be computed from scratch, but can be calculated very quickly from that of its predecessor. Suppose that the bytes between offsets b and e in F' have the form G1a...bG2, where the combined length of blocks G1 and G2 is |F| - 1, and a and b are single bytes. Then the candidate reconstruction G1aG2 will be followed by G1bG2, with a change in just one byte.

The CRC checksum of a bit-string b[n-1],...,b[0] is by definition the coefficient bit string of the remainder of the polynomial $\sum_{i=0}^{n-1} b_i x^{i+d}$ on division by a fixed irreducible degree-d polynomial P(x) over the integers modulo 2. Returning to bytes, linearity of the checksum implies that the exclusive-OR difference between successive candidates is:

$$\gamma \triangleq \mathbf{CRC}[\mathcal{G}_1\alpha\mathcal{G}_2] \oplus \mathbf{CRC}[\mathcal{G}_1\beta\mathcal{G}_2] = \mathbf{CRC}[(\alpha\oplus\beta), 0^{|G_2|}]$$

where 0**n represents a sequence of n bytes, each of value 0. Expressed as a polynomial (once more over bits), this difference is:

$$\gamma(x) = \sum_{j=0}^{7} \left[(\alpha_j \oplus \beta_j) x^{8|G_2|+j+d} \bmod P(x)\right] \bmod P(x).$$

Note that $\delta_b(x) \triangleq x^{b+d} \bmod P(x)$ can be computed from its predecessor, as $(x\delta_{b-1}(x)) \bmod P(x)$. This turns out to be a trivial computation in the bit-string representation:

$$\delta_k = \mathbf{LeftShift}[\delta_{k-1}] \oplus \mathbf{CRC}\,[\mathrm{MSB}(\delta_{k-1})]$$

where MSB represents the most significant bit, and the LeftShift operation shifts the bit string one position to the left, dropping the MSB from the left and bringing in a zero bit on the right. Since |G2| increases by one at each successive trial, gamma can be computed quickly, and in turn so can CRC(G1bG2) = CRC(G1bG2) XOR gamma. The computation of gamma may also be performed efficiently on bytes rather than bits, using shifts, exclusive-ORs, and a 256-element table of the CRC's of all byte values.

Rather remarkably, the generic disinfection algorithm can use a related trick to reconstruct programs infected with a limited class of overwriting viruses: viruses that replace some of the host with the virus to keep the host's length unchanged [6].

Generic disinfection performance The generic disinfection heuristic has an extremely low false positive rate. Since its incorporation into IBM AntiVirus in 1994, there have been no confirmed reports of a false positive. The false negative rate is also extremely low. In laboratory experiments, it caught over 99% of all non-overwriting virus infections. The proportion of viruses that overwrite their hosts is about 15%, but none of these are known to spread successfully because they are too harmful to go undetected for long. Thus in practice generic disinfection is extraordinarily effective. Of course, there is one important drawback: generic disinfection can only catch viruses in programs that have been seen in their uninfected state. But if a file-infecting virus spreads appreciably on a machine, the generic disinfection algorithm will almost certainly catch it.

Classifier A second method for detecting previously unknown viruses, the virus classifier, does not require prior knowledge of an uninfected version of a program. A classifier's function is to identify an object as either viral or non-viral; a classifier is constructed through a training procedure that takes as input a collection of known viruses and a collection of known non-viruses. The classification learning problem is hard in general and in this particular application; here the classification also needs to be accurate and robust.

Classifiers base their decisions on the presence or absence of various features, in this case short byte sequences common to several viruses. Candidate features are found using suffix arrays, a standard data structure for text algorithms [10]. The classifier itself is a simple hyperplane separator: the number of times each feature string occurs in a sample is tallied; the tallies are multiplied by weights and added together; and the sample is considered likely to be viral if the sum exceeds a threshold. The selection of features from the thousands of strings that occur repeatedly, and the determination of the feature weights, are particularly knotty problems: standard methods result in hopeless overtraining and abyssmal performance on non-training data. The techniques we used to overcome these obstacles will be described in a future publication, and have two patents pending.

The file-infector classifier contains roughly 1000 5-byte features. To elude virus scanners, some viruses use a simple form of encryption, taking the exclusive-OR of their own machine code with a repeated one- or two-byte key. To catch them, the features we use are not direct machine-code fragments, but transformations that are invariant with respect to such encryption. In particular, the 5-byte feature derived from the 7-byte fragment is its second-order exclusive-OR difference, . A file is classified by computing its invariant, tallying the number of times each feature occurs in the invariant, and computing a thresholded, weighted sum of the feature tallies.

Initial tests indicate that the classifier can detect roughly 85% of viruses on which it was not trained. However, with comprehensive false-positive tests still pending, we expect that to eliminate some false positives we will also have to reduce the detection rate somewhat. Still, by combining the generic disinfection and virus classification techniques, we believe that IBM AntiVirus will be able to detect approximately 99% of all new file infectors that actually spread in the wild.

3.1.2 Boot-sector infectors

A classifier for boot-sector infectors has also been implemented; in fact it came first. The boot-sector classifier [2] was trained separately on a corpus of several hundred legitimate and viral boot sectors.

This classifier uses roughly 30 three- and four-byte plaintext fragments. Although they were derived purely mechanically, the fragments turn out to be semantically meaningful bits of code. Many of them appear among a set of generic boot-sector signatures that had been hand-crafted by a human expert.

The boot-sector classifier's false-positive rate is quite low, but a few false positives have occurred. All have ultimately been attributed to special boot sectors that offer enhanced security for PC's, and that do several things that are virus-like. Since the "neural net" boot virus detector was first deployed in IBM AntiVirus in 1994, it has caught roughly 75% of all new boot sector viruses. Recent improvements include using an emulator to identify any additional relevant sectors, and decrypt them if necessary, prior to examination by the classifier. The improvements already made are believed to have increased the detection rate to roughly 90%, and further gains are expected.

3.1.3 Macro viruses

Currently, many new macro viruses are just simple variants of known ones. Consequently, fuzzy detection using simple signatures is sufficient to catch roughly two thirds of all new macro viruses. We are developing other techniques, including a classifier trained specifically on macro viruses, that are expected to bring the detection rate up to the high levels attained for file-infectors and boot-infectors. Rigorous false-positive testing is still required before the trade-off between minimizing false negatives and minimizing false positives can be tuned appropriately.

3.2 Capturing and transmitting virus samples

Whenever a heuristic identifies a potential virus, a sample is captured and compressed. Information about the type of virus, the version of the anti-virus program, etc., are included with the sample data, and the result is encrypted. The infected client PC then sends the encrypted sample to an administrative PC via a secure protocol that authenticates that the sample was generated by a legitimate version of IBM AntiVirus.

The administrator is given the opportunity to review the sample to ensure that it does not contain proprietary or sensitive data or code. Alternatively, the administrator may choose to configure the system to send samples outside the corporate firewall automatically. In either case, in the planned commercial version of the immune system, the sample is sent over the Internet to a concentrator located outside both the corporate and IBM firewalls, where other such samples from other administrative domains are collected. Another computer pulls the samples through the IBM firewall, and yet another through a final firewall and into the immune system laboratory.

There the sample is analyzed by the adaptive immune system, which is faced with the task of developing a specific response: means for detecting, verifying and removing all instances of the virus.

3.3 Deriving a prescription: the adaptive immune system

The adaptive component of the computer immune system can be viewed as an analog to the antibodies, B-cells, T-cells, and other elements of the adaptive biological immune system. In the current implementation, it does not reside on the client PC, for reasons of performance (the algorithms are CPU- and memory-intensive), convenience (we can focus development efforts on a single platform) and security (the algorithms require a database of infected files). [footnote 1]

When a sample is received, the address of the sender is recorded, and the sample and information pertaining to it and the environment in which it was found are uncompressed and decrypted. The sample is checked against the most current copy of the signature file because it is possible that the PC on which the virus was captured had a slightly out-of-date signature file. This step is particularly important for a rapidly spreading virus, about which many independent reports could be sent to the virus analyzer at nearly the same time. If the virus is known, the relevant anti-virus data can be sent back to the originally infected PC immediately. Otherwise, information about the type of virus is used to set up an appropriate environment for creating samples, or to route the sample to a machine where the appropriate environment can be found. So, file and boot virus samples are sent to an emulator (or to a real machine running the appropriate platform), Microsoft Word macro viruses to a WindowsNT environment running both Office 95 and Office 97,[footnote 2] etc.

Once the sample has been placed in an appropriate environment, it is encouraged to create additional samples of itself. If this succeeds, further analysis isolates code portions of the virus, which are useful for signature extraction and other purposes. Then, an "autosequence" procedure uses pattern matching to extract the virus's method of attachment and its basic structure. From this, a prescription for verifying and removing the virus is derived, and a virus signature is extracted. Finally, the virus data are tested, and integrated with data files that contain complete information for all known viruses. The resultant update is sent to the afflicted PC.

For the majority of viruses, the entire procedure takes less than two minutes from beginning to end on a dual 200MHz Pentium Pro PC running the Windows NT operating system. The remainder of this section treats its steps in greater detail.

3.3.1 Replication

The virus sample is placed in the appropriate environment and it is lured to infect a diverse suite of "decoy" or "goat" programs. For binary file infectors, there are several dozen .COM and .EXE-format decoys; for boot infectors, a few floppy and hard disk images; and for macro viruses, several documents or spreadsheets. Some viruses infect an object only when it is used, so the immune system executes, reads, writes to, copies, opens, and otherwise manipulates them.

Other viruses actively seek objects to infect, so decoys are placed where the most commonly used programs are typically located. For binary file infectors, this includes the root directory, the current directory, and other directories in the path. The next time the infected file is run, it is likely to select one of the decoys as its victim.

Periodically, each of the decoys is examined to see if it has been modified. If so, it is almost certain that an unknown virus is loose in the system, and that each of the modified decoys contains a sample. These virus samples are stored in such a way that they will not be executed accidentally.

Since viruses can be particular about the conditions under which they replicate, the decoys are run under several different environmental settings if necessary. The date may be changed a few times, as may the operating system version and other parameters. This helps to increase the chances of replication, and also helps expose differences in viral progeny that depend upon such parameters.

3.3.2 Behavioral analysis

For file and boot infectors, sample creation occurs in an emulated environment. The emulator produces a detailed log of system behavior. After the sample creation phase, the log is analyzed by an expert system that has been programmed with rules allowing it to determine various aspects of the virus's behavior [12,13]. For example, for file infectors the expert system is able to produce information about the types of executables the virus can infect, whether or not the virus "goes resident", and the conditions under which it infects its victims (e.g. the victim was just opened, closed, executed, etc.). Ultimately this information will be automatically incorporated into a "help" file describing the virus to customers and virus experts.

Replicating the virus in an emulated environment has the advantage of giving us data on the virus's behaviour, allowing us to automate the task, and keeping the virus isolated from the rest of the system while it is active. The disadvantage is the near-impossibility of performing high-quality PC emulation. Currently our emulator only runs code compatible with an 80386 in real mode. As a backup, we have another, hardware-based solution, but it is slower and has not yet been integrated into the automated system.

Macro viruses are replicated in an analogous manner. In the case of Word for Windows macro viruses, Word is placed in an armoured Windows NT environment. Word is driven by a script that loads the virus and attempts to infect various "decoy" or "goat" documents. After the virus has run, results are collected and the environment is restored.

3.3.3 Code/data segregation

Both the autosequencing and automatic signature extraction steps, to follow, need to know which portions of the virus are code and which are data. For file and boot infectors, we use an 80386 chip emulator that has been modified so that, when a conditional branch is encountered, all paths are taken. When the emulation is complete, portions of the virus that were executed as code are so identified, and the rest is tagged as data.

3.3.4 Autosequencing

Given several pairs of uninfected and infected decoy programs -- generated in the sample creation phase -- we use pattern-matching algorithms to extract a description of the virus's structure and of how it attaches itself to any host [14,15].

Pattern matching is first done within each file pair to determine the viral portions (strings occurring only in the infected file) and their locations. Then, the viral portions extracted from all the pairs are compared to identify patterns that are constant across all instances. (The comparison takes into account simple forms of encryption, as before.) Finally, an effort is made to locate portions of the original file that may have been sequestered within one of the variable regions of the virus. A simple example of the output from this stage is depicted in figure 1.

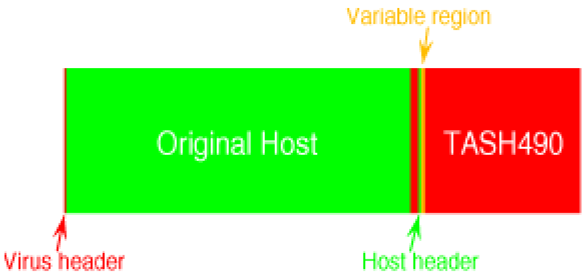

Figure 1: Pictorial representation of attachment pattern and structure of the TASH490 virus, derived completely automatically.

This "autosequencing" analysis provides several useful items of information:

1. The locations of all the pieces of the original host within an infected file, independent of the content and length of the host. This information is automatically converted into the repair language used by IBM AntiVirus.

2. The locations and structures of all components of the virus. Structural information includes the contents of regions of the virus that are observed to be invariant across different samples, and which have been classified as code in the code/data segregation step. This information has two purposes:

   (a) It is automatically converted into the verification language used by IBM AntiVirus.

   (b) It is passed to the automatic signature extraction component for further processing.

3.3.5 Automatic signature extraction

From among the byte sequences produced by the autosequencing step, the signature extractor must select a virus signature carefully to avoid both false negatives and false positives. That is, the signature must be found in every instance of the virus, and must almost never occur in uninfected programs.

First, consider the false negative problem. The samples captured by the decoys may not represent the full range of variable appearance of which the virus is capable. As a general rule, non-executable "data" portions of programs, which can include representations of numerical constants, character strings, and work areas for computations, are inherently more variable than are "code" portions, which represent machine instructions. To be conservative, "data" areas are excluded from consideration as possible signatures. Byte sequences that are both "invariant" according to the autosequencer, and "code" according to the code/data segregator, are the ideal input to the signature extractor.

The signature extractor is thus left with the problem of minimizing false positives. In the biological immune system, false positives that accidentally recognize self cause auto-immune diseases. In both traditional anti-virus software and the computer immune system, false positives are annoying to customers, and so infuriating to vendors of falsely-accused software that at least one lawsuit against a major anti-virus software vendor has resulted.

Briefly, the automatic signature extractor examines each subsequence of S contiguous bytes (referred to as "candidate signatures") within its input. and estimates the probability for that S-byte sequence to be found among normal, uninfected "self" programs. Typically, S is chosen to be between 16 and 24. The probability estimate is made by forming a list of all sequences of 5 bytes in the input data; tallying their frequencies in a corpus of roughly 20,000 ordinary, uninfected programs comprising a gigabyte of data; and using simple formulas based on a Markov model to combine the frequencies into a probability estimate that each candidate signature will be found in a set of programs statistically similar to the corpus. The signature with the lowest estimated false-positive probability is selected.

Characterizations of this method show that the probability estimates are poor on an absolute scale because code tends to be correlated on a scale longer than 5 bytes. However, the relative ordering of candidate signatures is good, so the method generally selects one of the best possible signatures [16]. The signature extractor has been used for several years now to generate signatures used by IBM AntiVirus. Judging from the relatively low false-positive rate of the signatures (compared with those of other anti-virus software), the algorithm's ability to select good signatures is better than can be achieved by typical human experts.

3.3.6 Testing and integration

The extracted signature is tested by verifying that it detects all the samples of the virus (including the original sample), and that it generates no false positives on the corpus of uninfected programs. The removal prescription is tested on each infected decoy to ensure that it produces a functionally equivalent version of the corresponding original decoy. Optionally, the sample creation step can be re-done with a fresh set of decoys, and both the signature and the removal information can be tested on them as well. If the anti-virus data pass the test, they are automatically integrated with the complete data file containing information for all known viruses.

3.4 Delivering the prescription

Finally, the data file update is sent back to the client from which the virus sample was received. In the current prototype, the update is sent directly to the client machine. In the commercial version, it will retrace the path by which it received: it will be pushed through the immune system laboratory firewall, pushed through the IBM firewall to the concentrator, pulled through the client company's firewall to its administrative machine, and finally delivered to the afflicted PC.

3.5 Disseminating the prescription

Having received the prescription, the administrator will (automatically) disseminate the cure to all machines that might need it, including the one on which the virus was originally found.

In the current prototype and in the planned commercial release, each PC runs IBM AntiVirus in the background upon receiving the update. All copies of the virus lurking in the system are detected, and the user is given the option to disinfect.

In the commercial version, the master virus data files in the immune system laboratory would also be mirrored at the "antivirus online" web site, www.av.ibm.com. Thus subscribers all over the world would be protected against new viruses soon after their discovery, provided that they download the updates frequently.

4. Evaluation and final remarks

Now we return to the criteria set forth in section 2, reproduced here in italics, and discuss the extent to which the immune system described in section 3 meets each of them.

- **Innate immunity.** *The system must be capable of detecting the presence of a high proportion of viruses that are unknown to it specifically. The system should recognize previously unknown viruses of all types; currently, this would include file infectors, boot-sector infectors, and macro viruses. It is particularly important that the system be geared towards detecting viruses that are found in real incidents. For example, less emphasis need be placed on detecting overwriting viruses, since these are never prevalent.*

  The discovery rate is roughly 99% for file infectors and 90% for boot-sector infectors. The figure for file infectors is averaged over non-overwriting viruses, which we feel is legitimate because overwriters have never successfully spread in the wild. Currently, the discovery rate is an unacceptably low 65% for macro viruses, but several techniques under development are expected to boost the discovery rate to at least 90% by the time the pilot is launched later this year.

- **Adaptive immunity.** *The system must be capable of automatically deriving from a single sample a "prescription" for detecting and, if possible, removing all instances of the virus.*

  The adaptive immune system prototype can usually produce an adequate supply of samples from a single sample of a file infector or a macro virus. (Less emphasis has been placed on boot-sector replication since such viruses are much less likely to take advantage of the Internet to spread quickly; still, this is not ideal.) In cases where samples are produced, a signature can be extracted in almost every case, even for encrypted viruses. For file infectors, automatic extraction of repair information can be tricky, and the success rate is lower than for signature extraction. For macro viruses, the success rate for extraction of both signatures and repair information is extremely high.

- **Delivery and dissemination.** *The system must deliver the prescription to the afflicted computer, and must also facilitate dissemination of the prescription to other machines both in its neighborhood and around the world.*

  In the prototype, the prescription is delivered directly from the adaptive immune system to the afflicted PC. In the commercial version still being imlemented, the prescription is delivered to the afflicted PC and disseminated in its locale by the administrative component of IBM AntiVirus, and made available to the rest of the world via updates downloadable from www.av.ibm.com.

- **Speed.** *The system must be capable of deriving and disseminating prescriptions on a time scale faster than the virus can spread. Currently, a reasonable goal is to derive and deliver a prescription to the originally afflicted machine in less than ten minutes from when an anomaly is first detected, and to permit dissemination of the prescription to neighboring machines within less than half an hour. The system should be capable of protecting anyone from any newly discovered virus within a day at most. These requirements will become even stricter as technology permits computer viruses to spread faster.*

  The immune system prototype easily satisfies the speed requirements. Once the sample is received, it typically takes less than two minutes to derive a full prescription for detecting and removing a file infecting virus, and roughly five minutes for a macro virus. Even if one allows a few minutes for the sample to be delivered via the network to IBM and for the prescription to be delivered over the network from IBM, the criterion for rapid delivery should be met comfortably. Dissemination of the prescription to other machines in the vicinity of the one that discovered the virus will be achieved by simple extensions to the existing IBM AntiVirus administrative software, which permits administrators to update hundreds or thousands of IBM AntiVirus clients within an administrative domain. Even by conservative estimates, it would take less than half an hour to send updates to, say, 1000 clients. Finally, by incorporating the prescription into the IBM AntiVirus data files and making them available from the web, a very large number of customers can be protected from the new virus within hours of its first discovery.

- **Scalability:**

  Analysis. *If a new, particular virulent virus begins to spread quickly, many machines at one site and/or at different sites may experience the virus simultaneously. The system must be capable of very high throughput, handling at least thousands of simultaneous requests for virus analysis.*

  The commercial pilot will be able to handle a limited number of multiple requests simultaneously. A proprietary design, slated for incorporation into the full commercial version of the immune system next year, will be able to handle a high volume of simultaneous requests.

  Updates. *The system must be able to update tens of millions of PCs on at least a daily basis.*

  A large number of customers may download up-to-the-minute updates from the Web, enabling them to be protected very soon after a new virus is discovered.

- **Safety and reliability.** *The prescription produced by the immune system must be sufficiently reliable that it can be deployed worldwide without being checked by humans. Preferably the false positive rate should be at least as low and the removal information at least as accurate as is typically achieved by expert humans.*

Tests suggest that the automatic signature extraction algorithm selects signatures that are less prone to false positives than those chosen by human experts [16]. The removal information extracted by the autosequence algorithm tends to be of high quality. This, coupled with the fact that IBM AntiVirus verifies the exact identity of a virus prior to removing it from a host program, means that botched repairs based an automatically generated prescription are very rare. Certainly, the rate is very much lower than the figure of 10% we have measured for a competing anti-virus product that presumably uses prescriptions derived by human experts!

- **Security.** *Virus samples must be protected in such a way that they cannot be intercepted and read by a third party while in transit. Even more importantly, appropriate measures must be taken to ensure that any prescriptions employed by a client machine are uncorrupted, genuine copies of those produced by the immune system.*

  Standard secure protocols are included in the design of the immune system. Encryption is used to ensure that virus samples cannot be intercepted by a third party, and digital signatures used to verify the origin of virus samples, anti-virus prescriptions, and updates.

- **Customer control.** *Customers must be able to control the conditions under which virus samples and other information are sent out of their enterprise. They must likewise be able to control the conditions under which prescriptions are received and disseminated to the originally infected client and possibly other machines. Some customers may want manual control in one or both directions, others may want the speed given by completely automated response.*

  Especially when the immune system is first made available commercially, administrators may wish to review the virus samples that are sent to IBM for automatic analysis to ensure that confidential material is not inadvertently included. Manual throttling of the flow of samples to IBM will always be an option, but as mechanisms that reduce such concerns are developed and implemented, administrators may permit the samples to be sent automatically. Likewise, manual control over the delivery of the prescription to the afflicted PC and other machines on the network must always remain an option, but we anticipate that with time it will become increasingly common to let updates occur automatically with no intervention from the administrator.

In summary, the existing prototype meets many but not all of the criteria. As we ready ourselves for the first commercial pilot in 1997, our primary emphases include implementing and testing our algorithms for boosting the detection rate of new macro viruses, finishing the implementation of the communication protocols between the IBM AntiVirus administrative and client components, and completing implementation of the network software that transfers virus samples and anti-virus prescriptions between IBM and the IBM AntiVirus Administrator program.

The IBM immune system is the product of several years of automation efforts. So far, six U.S. patents relating to it and its components have been granted [17,18,15,6,7,19].

Acknowledgments

The authors thank their many colleagues at IBM's High Integrity Computing Laboratory, particularly Bill Arnold, Dave Chess, and Ed Pring, for numerous discussions about automatic virus analysis and the design of the immune system. Bill Arnold and Dave Chess invented the idea of using decoy programs. John Evanson, Riad Souissi, Bill Schneider, Frederic Perriot, Jean-Michel Boulay, Hooman Vassef, and August Petrillo contributed to the development of the adaptive immune system, and Alexandre Morin and Gerald Tesauro made major contributions to the virus classifier.

Footnotes

[footnote 1] Eventually, it could be advantageous to decentralize virus analysis, even to the point of having the innate and adaptive immune systems together on each PC. This would be truer to the biological analogy, but more importantly it could be helpful for different computers to use different signatures to detect the same virus. For an interesting treatment of the virtues of diversity in computer systems, see Forrest et al. [11].

[footnote 2] Microsoft changed the underlying macro language used in their Office application suite between these versions. However, the new language (VBA5) is backwards compatible and the viruses must be replicated on both platforms.

References

[1] J.O. Kephart. A biologically inspired immune system for computers, in R. A. Brooks and P. Maes, eds., Artificial Life IV. Proc. of the 4th International Workshop on the Synthesis and Simulation of Living Systems, 130-139. MIT Press 1994.

[2] J.O. Kephart et al. Biologically inspired defenses against computer viruses, Proceedings of IJCAI '95, 985-996, Montreal, August 19-25, 1995.

[3] C.A. Janeway, Jr. How the immune system recognizes invaders. Scientific American, 269(3):72-79, September 1993.

[4] P. Marrack and J.W. Kappler. How the immune system recognizes the body. Scientific American, 269 (3):81-89, September 1993.

[5] Paul, William E., ed. Immunology: Recognition and Response : : :Readings from Scientific American. New York: W.H. Freeman and Company, 1991.

[6] J.O. Kephart and G.B. Sorkin. Generic disinfection of programs infected with a computer virus. U.S. Patent 5,613,002; March 1997.

[7] J.O. Kephart, G.B. Sorkin and G.J. Tesauro. Adaptive statistical regression and classification of computational bit strings : : :U.S. Patent (to be issued); allowed March 1997.

[8] D. Bertsekas and R. Gallager. Data Networks, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[9] R.M. Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development, 31:249-260, 1987.

[10] M. Crochemore and W. Rytter. Text Algorithms, Oxford University Press, New York, 1994.

[11] S. Forrest, A. Somayaji, and D.H. Ackley. Building Diverse Computer Systems. In Proceedings of 6th Workshop on Hot Topics in Operating Systems, 67-72, IEEE Computer Society Press, Los Alamitos, CA, 1997.

[12] M. Swimmer. Dynamic detection and classification of computer viruses using general behavior patterns. In Proceedings of the Fifth International Virus Bulletin Conference, 75-88. Virus Bulletin Ltd., 1995.

[13] M. Swimmer. Fortschrittliche Virus-Analyse -- Die Benutzung von statischer und dynamischer Programm-Analyse zur Bestimmung von Virus-Charakteristika. Master's Thesis, Fachbereich Informatik, Universitaet Hamburg, May 1995.

[14] W.C. Arnold and G.B. Sorkin. Automated analysis of computer viruses. In Proceedings of the Sixth International Virus Bulletin Conference, 149-159. Virus Bulletin Ltd., 1996.

[15] D.M. Chess, J.O. Kephart and G.B. Sorkin. Automatic analysis of a computer virus's structure and means of attachment to its hosts. U.S. Patent 5,485,575; September 1995.

[16] J.O. Kephart and W.C. Arnold. Automatic extraction of computer virus signatures. In Proceedings of the Fourth International Virus Bulletin Conference, 179-194. Virus Bulletin Ltd., 1994.

[17] W.C. Arnold, D.M. Chess, J.O. Kephart, and S.R. White. Automatic immune system for computers and

computer networks. U.S. Patent 5,440,723, August 1995.

[18] J.O. Kephart. Method and apparatus for evaluating and extracting signatures of computer viruses and other undesirable software entities. U.S. Patent 5,452,442; September 1995.

[19] W.C. Arnold, D.M. Chess, J.O. Kephart, G.B. Sorkin and S.R. White. Searching for patterns in encrypted data. U.S. Patent 5,442,699. August 1995.

back to index